INVESTIGATING COMMUNITY DETECTION ALGORITHMS FOR MEETING SUMMARIZATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Shantanu Gore

May 2019

ABSTRACT

We extend a fully unsupervised, abstractive meeting summarization framework to use novel clustering methods. We investigate the application of the Word Mover's Distance and variants of it, as well as various clustering methods such as agglomerative clustering, spectral clustering, and k-means applied to data generated using multidimensional scaling. Our embedding-based distance approach encorporates exterior knowledge into the clustering stage of the framework.

BIOGRAPHICAL SKETCH

Shantanu Gore was born in Thane, India and grew up in Herndon, Virginia. After graduating from Thomas Jefferson High School for Science and Technology in 2015, he entered Cornell University to study Computer Science. He earned his BS in Computer Science with Honors in December 2017 and graduated *summa cum laude*. He earned his MS in Computer Science from Cornell University in May 2019.

This document is dedicated to my parents, Anuprita and Devidas Gore.

ACKNOWLEDGEMENTS

We would like to thank Professor Claire Cardie for all the support and guidance in conducting this research. We are also grateful to Professor Siddhartha Banerjee and the Cornell University Computing and Information Science department.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

As the amount of available knowledge increases in the era of the internet, automated summarization has become an increasingly important task. Formally, the task of automated summarization is to find the most relevant information in text document(s) and condense it. The summaries generated by an automated system should contain the important information in the document(s) they are summarizing, and be coherent, grammatical, and non-redundant.

Current approaches to summarization can be largely divided into two parts. One approach, known as *extractive summarization* focuses on including segments from the source document(s) (usually at the granularity of sentences or sentence fragments) to include in the summary, and concatenates these segments together to form a summary. *Abstractive summarization*, the other main approach, uses natural language models to generate a summary given some source document(s). Specifically, this leads to summaries that contain words or phrases not found in the source(s), and as such can produce more coherent summaries. Commonly used abstractive summarization techniques include sentence comprehension, syntactic reorganization, and lexical paraphrasing.

A subfield of interest of automated summarization is that of *multi-document summarization*, in which systems are presented with multiple related documents from which they create a single summary (for example, multiple news articles about an event may be used to create a summary). One can imagine approaching this task as generating separate summaries for each input document and concatenating

them, but this task is more challenging than single document summarization due to information overlap. Specifically, such a system that simply summarized each input separately would generate a summary with redundancy, due to redundancy across the input documents.

A commonly used solution to this problem is multi-sentence compression, in which related sentences are grouped together and summarized together. This is essentially transforming an input set of documents D into another set of documents D', where each document $d \in D'$ is about a single topic. Note that the approach of summarizing each document separately and then combining the summaries works better on D', as there would be less overlap across the documents in D'. In fact, we can use this technique in the single-document summarization task as well - given a single document d, we can partition it a set of documents D' such that each document $d' \in D'$ is about a single topic.

Meeting summarization is an interesting application of automated text summarization, in which the input is the transcript of a meeting. This task presents more difficulties than standard document summarization, due to the data being sourced from speech. Specifically, the data is riddled with incomplete sentences, irrelevant conversation between participants (eg. `How was your weekend?` and `Mine was great, how about yours?`, etc.), and redundancy induced by multiple participants agreeing on an idea. Additionally, we note that the input, created using automated speech recognition (ASR) on a recording of a meeting, is also noisy due to the imperfection of ASR. Finally, we note that this task is actually quite similar to the multiple document summarization task, as although there is only one document we are summarizing, it is composed by multiple speaker (meeting

participants). As such, we face the same challenges as in solving that problem, namely redundancy.

In this paper, we investigate modifications to the community detection phase to the framework proposed by [1]. Specifically, we investigate new distance functions and clustering methods. The rest of this paper is organized as follows: in Section 2 we present a review of the existing literature, in Section 3 we discuss the modifications we propose to the existing framework. In Section 4 we summarize the experiments performed on our systems to evaluate their performance, and in Section 5 we discuss the results of these experiments. We conclude in Section 6 and discuss some potential directions for future work.

CHAPTER 2

LITERATURE REVIEW

In [2], Filippova et al. propose a novel approach to summarization they call *multi-sentence compression* (MSC). Specifically, their algorithm takes as input a group of related sentences $S = \{s_1, ..., s_n\}$, and outputs a single short sentence that summarizes the group. They do so by introducing a new construct called a *word graph*, which they define as a directed graph where an edge between word A and B indicates that the bigram AB occurs in a sentence $s \in S$. They also add artificial nodes labeled `start` and `end` to this graph, and connect `start` to the first word of each sentence $s \in S$, and connect the last word of each sentence to `end`. They note that this approach does not specifically model grammaticality, but propose that redundancy (among multiple sentences in S) provides the ability to identify both important words and important links between words. Further, we note that this is quite similar to a bigram model, and as such the links themselves can be thought of as inherently modeling grammaticality. Their algorithm builds the word graph by iteratively adding each sentence $s_i \in S$ to it. Specifically, given a group of sentences S, their algorithm chooses one of the sentences s_1 and creates nodes for each of its words, with each edge having a weight of one. After the sentence is added, the algorithm considers each successive sentence in S, and tries to map its words onto nodes in the graph (that meet certain conditions, such as the two words having the same lower cased form, same part of speech, etc.) and increment the edge weight, or creates a new node if no such node exists. These rules for merging nodes create a graph satisfying the following desirable properties, which reduce the chances of creating ungrammatical sentence in the next part of the algorithm:

1. Every input sentence corresponds to a loop-less path in the graph
2. Words referring to same entities or actions are likely to be mapped to the same node
3. Stopwords are only combined in one node if there is an overlap in context

In order to generate grammatical summary sentences, Filippova et al. use graph algorithms on the created word graph. Specifically, they note that the summary sentences should satisfy the following key properties:

- Appropriate length - not too long, but also not so short as to not be meaningful
- Cover important concepts, but not be repetitive
- Grammatical

To achieve these goals they first invert the edge weights (previously frequencies) to be $\frac{1}{\text{frequency}}$, so that frequently used edges now have small weights. They then find the shortest path from the inserted `start` node to the inserted `end` node. They generate the K shortest paths, and do some filtering on the selected paths to ensure that they are at least a preset length and have a verb node.

They also propose another more sophisticated method, which involves two major factors: strong links, to ensure grammaticality, and salient nodes, to ensure informativeness. Specifically, they want the generated sentences to use a sequence of words in which pairs of words are strongly associated with each other. They

note that simple inverse frequency does not measure this well, because it does not consider how often the individual words occur in S, only how often they occur consecutively. Furthermore, they note that information can be extracted from the repeated cooccurrence of words, even if they are not consecutive. For this reason, they redefine edge weight to be

$$w(e_{i,j}) = \frac{\text{freq}(i) + \text{freq}(j)}{\sum_{s \in S} \text{diff}(s, i, j)}$$

where $\text{diff}(s, i, j)$ is the minimum distance between words i and j in sentence s, or 0 if s does not contain both i and j. This gives preference to edges between uncommon words over edges between common ones. Next, they note that the above weight function only considers how frequently words occur together (words that occur consecutively in one document vs. in every document would have the same weight). They note that it is important for paths to go through "salient nodes", or nodes that occur in multiple sentences, and modify edge weights to be:

$$w(e_{i,j}) = \frac{\text{freq}(i) + \text{freq}(j)}{\left(\sum_{s \in S} \text{diff}(s, i, j)\right) \times (\text{freq}(i) \cdot \text{freq}(j))} \tag{2.1}$$

Using these updated edge weights, they then generate K shortest paths in the document. Noting that longer paths would have higher weights simply due to their length, they then renormalize the total weight of each path by the number of words it contains, and choose the path with the lowest average edge weight.

An example of this algorithm being applied is shown in Figure 2.1. Specifically, we create a word graph with the following sentences, noting that for clarity edge

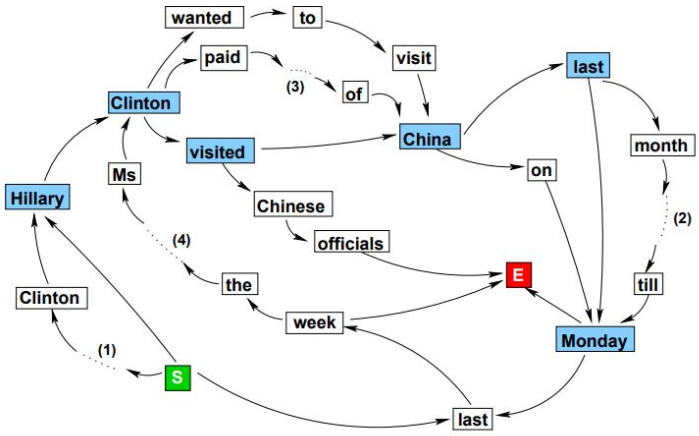

Figure 2.1: Multi-sentence compression [2].

weights have been omitted, and the italicized parts of each sentence are replaced with dots in the graph:

- *The wife of a former U.S. president Bill* Clinton Hillary visited China last Monday.
- Hillary Clinton wanted to visit China last month *but postponed her plans* till Monday last week.
- Hillary Clinton paid *a visit to the Peoples Republic of* China on Monday.
- Last week the *Secretary of State* Ms. Clinton visited Chinese officials.

We note that just by using short paths in the graph, the algorithm is able to effectively summarize these sentences into "Hillary Clinton visited China last Monday". Furthermore, we also note the importance of the minimum path length constraint, as there do exist shorter paths in this graph, such as "last week", but clearly these do not contain enough information to be useful.

In [3], Boudin and Morin propose an improvement to [2]'s MSC model, specifically by modifying the path re-ranking algorithm. Firstly, in order to produce more grammatical sentences, [3] first extend [2]'s model by inserting punctuation marks into the graph, following similar rules as for the words in the sentences. Recall that the method proposed by [2] picked the K shortest paths, with edge weights as defined by Equation (2.1). After selecting these paths, they re-ranked them based on the average edge weight, defined as the path cost divided by the number of nodes contained. In their paper, Boudin and Morin note that the key assumption behind [2]'s approach is that redundancy in the set of sentences $S = \{s_1, ..., s_n\}$ would provide a reliable way to generate informative and reliable sentences. However, in practice, they find that around 50% of the sentences produced by [2] are missing important information, and they propose a method to produce more informative sentences by directly maximizing the range of topics covered, rather than counting on redundancy to handle this implicitly.

Their main contribution involves an improved way (over occurrence count) to measure node salience using keyphrase extraction, a common task in NLP, involving capturing the main topics of a document. They claim that an informative sentence (again using the same two metrics as [2], grammaticality and informativeness) should contain the most relevant keyphrases, and propose the count of keyphrases in a summary sentence as a method for re-ranking. They use a model by [4] for keyphrase extraction, using a recommendation-based algorithm - words w_i recommend neighboring words w_j as being important, and the strength of these recommendations is related to the importance of w_i.

Their method for keyphrase extraction consists of first making a graph of related

sentences S with nodes being tuples of words and their part of speech (POS) tag. An edge connects nodes i and j if they cooccur in the same sentence $s \in S$, with the weight being the number of sentences s_i that contain both i and j. The TextRank algorithm [5] is then applied to this graph to compute a *salience score*, S_i for each node, defined as (where w_i and w_j are *adjacent* if they are neighbors in the constructed graph)

$$S(w_i) = (1-d) + d \cdot \sum_{w_j \text{ adjacent } w_i} \frac{w_{j,i}}{\sum_{w_k \text{ adjacent } w_i} w_{j,k}} S(w_j)$$

After each node is assigned a saliency score, keyphrase candidates are generated and scored. Specifically, text segments meeting certain POS criteria (eg. `(ADJ) * (NPP | CC)`, ...) are collapsed into multi-word phrases, and the score of a phrase k is computed as the sum of its nodes saliency scores normalized by its length `+ 1`. The authors of [3] note that this task is made relatively computationally easy due to the small vocabulary size within S and high redundancy of the sentences. However, this also leads to redundant keyphrases and keyphrases containing other ones, such as `fire truck` and `rubber fire truck wheels`. This is addressed by clustering generated keyphrases using word overlap, and then choosing the keyphrase with the highest score from each cluster.

Additionally, due to the potentially large variance between raw path length (as defined by the weights in equation 2.1) and the best path in terms of informativeness (defined using keyphrases), K is set higher than the value in [2]. Each of the K shortest paths p generated is then re-ranked by (where p is the path being scored,

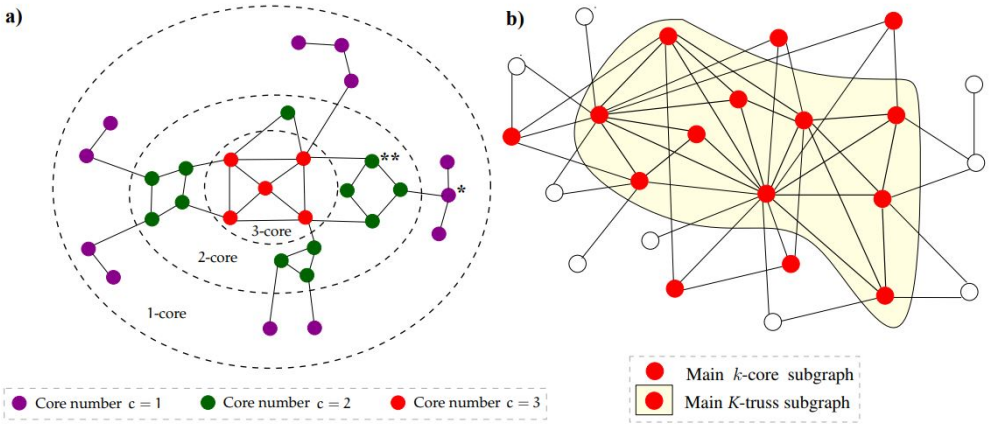

Figure 2.2: Figure 2.2a) shows an example k-core decomposition. Figure 2.2b) illustrates the difference between a k-core and a K-truss [6].

and $k \in p$ are keyphrases contained in p):

$$score(p) = \frac{\sum_{(i,j) \in p} w_{i,j}}{\text{length}(p) \cdot \sum_{k \in p} \text{score}(k)} \tag{2.2}$$

In [6], Tixier et al. propose a novel approach to keyword extraction. Specifically, they note that while previous work has used spectral (eg. eigenvector-based) centrality measures on graphs of words to find keywords (which tend to find *prestigious* nodes), they are more likely to be *influential* nodes of this graph. This means that these nodes may not necessarily have many important direct connections, but they are placed at the core of the global graph - this is essentially a shift from local attributes, such as quality and quantity of single node connections, to global attributes such as density and cohesiveness of groups of nodes.

They start by creating a graph-of-words G for a document, as presented by [7], in which a node is created for each unique noun and adjective in a document, with edges created between words if they occur with a certain span W of each

other, including across sentence boundaries. Each edge is assigned a cooccurrence count, representing the number of times these words occur within the window W of each other. They then define the k-core subgraph of G as the *maximal connected subgraph of G in which each vertex has degree at least k*. They further define the k-core decomposition as the set of all cores from the 0-core (G itself) to $k_{\max}$-core, and the *core number* of a node to be the highest core in which it appears. In Figure 2.2a) we see an example of a k-core decomposition. We note that the two nodes labeled by (*) and (**) both have degree 3, but the (**) node has a higher core number. This correlates well with its more central position in the graph. They also introduce the notion of a k-truss [8], which is an edge-based extension of k-core, pruning nodes based on shared connections rather than direct links, and a k-shell, which is the set of all nodes with core number k. Figure 2.2b) shows the difference between the k-core and the K-truss of a graph. Tixier et al. note that "the main K-truss subgraph can be considered as the *core* of the main core."

Tixier et al. leverage a result from social network theory, which states that the best *spreaders* in a network (those nodes best able to propagate information to a large portion of the network in minimal time and with minimal cost) are those located at the core of the network, rather than those that are highly connected [9]. For this reason, they claim that the words at the core of the constructed graph-of-words G will be influential words, and thus good candidates for keywords. Next, they note that while a high core number is a good indicator of a keyword, all keywords may not have a core number of $k_{\max}$, and thus it is necessary to consider non-maximal cores as well. As such, some criterion is needed for selecting the core number at which to search for keywords.

They define the *density* of a graph to be

$$\text{density}(G) = \frac{|E|}{|V|(|V|-1)}$$

which is proportional to the ratio of the number of edges it contains to the number of edges contained by a clique on the same number of nodes. They then compute the density of each core, and choose a value based on the combination of the core number and the density value. They also define an *inflection* based metric, which essentially chooses the smallest k such that the $k+1$th shell is smaller than the kth shell. Finally, they also present the *CoreRank* metric, which has the advantage of being able to choose certain nodes in a shell while ignoring others. This is clearly beneficial, as we would not expect all nodes in a shell to be keywords. Specifically, they define a score (CoreRank, CR) for each node, which is the sum of the core numbers for all of its neighbors. Note that unlike the salience score from the TextRank algorithm, this is not a recursively defined function and thus is efficient to compute (does not require iterative methods):

$$\text{CR}(v) = \sum_{u \in N(v)} number(u)$$

and rank nodes based on this score (where $N(v)$ are the neighbors of node v in the present subgraph). This has the benefit of operating at a node level granularity rather than a set-of-nodes (k-shell) level granularity. The algorithm then selects either the top $p\%$ of nodes as keywords, or an optimal value considering both the

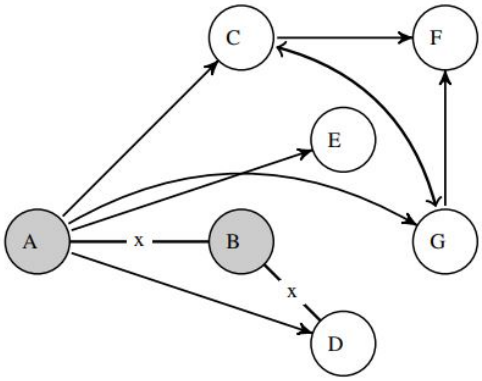

Figure 2.3: Example entailment graph [10].

core number and the CR score. This method also serves to stabilize the CR scores in a neighborhood of the graph, which provides more robustness against noise, especially relevant when working on an automated speech recognition (ASR) based corpus.

In [10], Mehdad et al. propose a supervised end-to-end framework for abstractive meeting summarization. Their system involves a multi-step pipeline, consisting of first clustering in the input into communities, building an *entailment graph* for each community to identify the most relevant sentences, aggregating these sentences using a word graph, and finally creating a summary sentence by selecting a path in the constructed word graph. The make three major contributions: a full pipeline to generate an abstractive summary of a meeting, a modification of [2]'s word graph, an entailment graph over sentences to identify important sentences, and a robust method for generating grammatical, well-formed summary sentences from poorly formed input data (such as that created from ASR transcripts).

As training data, [10]'s model requires a gold-standard summary, along with tags

for which sentences in the meeting contributed to each part of the summary. Part of their generation algorithm involves identifying which sentences should be combined (or "fused") to form a summary sentence. In order to find these sentences, they first perform a task introduced by [11]: deciding for each pair of sentences s_1, s_2 if they should be summarized by a single abstractive sentence. This is done using a logistic regression classifier on the sentences' linguistic and structural features, and the outputs are used to create an undirected graph G containing all the sentences in the document. The CONGA algorithm [12] is used to find communities in this graph, with the betweenness metric for each edge $e \in E(G)$ defined as the number of shortest paths in the graph that pass through e.

Semantic information about the sentences in each community is then captured in the form of an entailment graph. Edges are constructed between each pair of sentences (s_1, s_2), annotated with one of three possible relations: bidirectional entailment (the sentences capture the same information, so one should be removed), unidirectional entailment (s_1 captures more information than s_2, so s_2 should be removed), or unknown entailment (s_1 and s_2 capture different information, so they should both be kept). These relations are computed using an SVM, using an 18-dimensional feature vector where each element is specific a similarity score that estimates whether s_1 entails s_2. These features include syntactic metrics such as edit distance and n-gram overlap, as well as semantic relations such as word synonymy, etc. Since the SVM only predicts directional entailment, it is run twice and the two results are aggregated into one of the three relations. After all relations are computed, they are compacted with the following rules: if two nodes are connected with bidirectional entailment, the one with more outgoing bidirectional and unidirectional entailments is kept, and all nodes in a chain of

entailing nodes are removed except for the root. Figure 2.3 shows an example of an entailment graph, with unidirectional arrows denoting unidirectional entailment, bidirectional arrows denoting bidirectional entailment, and lines with `x` denote unknown entailment.

The remaining sentences in each community are then used to create a word graph. This is similar to that of [2], with a few modifications. Mehdad et al.'s system uses WordNet to combine semantically similar nodes, allowing for sentences to include words not in the input sentences (thus making the model more abstractive). Similar to the approaches by [2] and [3], the K shortest paths in the word graph are constructed, and paths not including a verb are pruned to help achieve grammaticality.

They note that a high quality summary sentence (denoted by a path P) for a community has the following characteristics: *fluency*, which is estimated using a standard n-gram language model:

$$\begin{aligned}\text{fluency}(P) &= Pr(P) \\ &= \prod_{i=1}^{m} Pr(p_i|p_1^{i-1}) \\ &\approx -\sum_{i=1}^{m} \log Pr(p_i|p_{i-n+1}^{i-1})\end{aligned}$$

coverage, which is estimated using the occurrence of salient nouns (measured using

tf-idf score, where $n \in P$ denotes nouns in the summary and $n \in G$ denotes all nouns in the community):

$$\text{coverage}(P) = \frac{\sum_{n \in P} tfidf(n)}{\sum_{n \in G} tfidf(n)}$$

and low edge weight $W(P)$, computed identically as in Filippova's paper (Equation 2.1). These metrics are combined to form the score for each path as follows, and the path with the highest score is chosen as the summary sentence:

$$\text{score}(P) = \frac{Pr(P) \cdot \text{coverage}(P)}{W(P)}$$

In [13], Lin and Bilmes use submodular optimization as a method of document summarization. This method comes with all the benefits of submodular optimization, including a simple greedy algorithm that is guaranteed to arrive at a solution $\hat{S}$ almost as good as the optimal solution S_{opt}. This is in fact a constant-factor approximation - letting $\mathcal{F}$ be an objective function, we have the guarantee that:

$$\mathcal{F}(\hat{S}) \geq (1 - \frac{1}{e})\mathcal{F}(S_{opt})$$
$$\approx 0.632\mathcal{F}(s_{opt})$$

Since this bound is multiplicative (the ratio does not depend on the value of $\mathcal{F}(S_{opt})$), it is especially attractive for the problem of summarization, since these problems can be quite large. Furthermore, Lin and Bilmes note that this is a worst case bound, and in practice many of the results produced by submodular optimization will be much better than the bound guarantees. The authors note that in order for this to be useful, however, we need an objective function that works well for summarization and is also meets the desired mathematical requirements: submodularity and monotonicity. The authors note that in many prior works in summarization, the objective functions used have actually been submodular, although this was not explicitly noted in those works. They claim that this demonstrates that submodular functions are a natural choice of objectives for summarization tasks.

Consider a set of sentences $D = \{s_1, ..., s_n\}$ and a function $\mathcal{F} : 2^D \rightarrow \mathbb{R}$ that scores a subset of sentences S from D. Summarization is often defined as choosing a subset S such that $|S| \leq k$ that maximizes $\mathcal{F}(S)$. We note that the naive solution to this optimization problem is, of course, computationally intractable as it scales exponentially with the number of sentences in D. We note that if the function $\mathcal{F}$ is submodular the exact optimization problem is still NP-complete, but there exists a greedy algorithm that is guaranteed to produce a solution within approximately

0.63 of the optimal solution [14]. We note that submodularity encompasses the idea of *diminishing returns* - a function $\mathcal{F}$ is *submodular* if

$$\text{for any } A \subseteq B \subseteq D \setminus v, \mathcal{F}(A+v) - \mathcal{F}(A) \geq \mathcal{F}(B+v) - \mathcal{F}(B)$$

Further, we note that a function $\mathcal{F}$ is *monotone nondecreasing* if

$$\text{for any } A \subseteq B, \mathcal{F}(A) \leq \mathcal{F}(B)$$

Lin and Bilmes also make note of the following theorem regarding composition of submodular and concave functions:

Theorem. *Given functions $\mathcal{F} : 2^D \to \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$, the composition $\mathcal{F}' = f \circ \mathcal{F} : 2^D \to \mathbb{R}$ is monotone nondecreasing and submodular if f is nondecreasing concave and $\mathcal{F}$ is monotone nondecreasing submodular.*

Having given an overview of submodularity, Lin and Balmes then go on to discuss applications of submodularity specifically to summarization. We note that earlier we defined a set-size constraint on the generated summary S, but in reality we often would like a length-based constraint. As such, they define costs c_i for

each $s \in D$ (such as word count or number of characters), and reformulate the constraint as $\sum_{i \in S} c_i \leq b$, where b is our *budget*. The summarization problem is thus defined as finding $S^* \in \text{argmax}_{S \subseteq D} \mathcal{F}(S)$ subject to $\sum_{i \in S} c_i \leq b$. The point out that a similar, previously proposed, summarization method known as *maximal marginal relevance* [15], which greedily selected sentences that are relevant while also removing redundant sentences, also satisfies submodularity.

The authors go on to present another formulation of the summarization problem, this time approaching the problem from the opposite direction - covering a document (i.e. contain all or some portion of the information in the document) while minimizing summary cost: $S^* \in \text{argmin}_{S \subseteq D} \sum_{i \in S} c_i$ subject to $\mathcal{F}(S) \geq \alpha.$, where $\mathcal{F}(S)$ now measures the information covered by S.

Having shown that choosing submodular objectives allows for an efficient approximation algorithm, Lin and Balmes proceed to actually present examples of such objective functions. They note that prior work often encouraged relevancy and penalized redundancy, which is not monotonic (due to the redundancy penalty). They instead positively reward diversity, and thus create the following objective function:

$$\mathcal{F}(S) = \mathcal{L}(S) + \lambda \mathcal{R}(S)$$

where $\mathcal{L}$ measures coverage of the summary, λ is a tradeoff factor, and $\mathcal{R}(S)$

measures the diversity of the summary. It is clear that $\mathcal{L}(S)$ is monotone, as adding information to a summary can only increase its coverage. Furthermore, this is also submodular, as when adding information to a larger summary, it may have already been covered by the existing sentences in the summary. They note that several common coverage functions can be used, they choose to use:

$$L(S) = \sum_{i \in V} \min\{\mathcal{C}_i(S), \alpha \mathcal{C}_i(V)\}$$

Here, $\mathcal{C}_i : 2^V \rightarrow \mathbb{R}$ is a monotone submodular function, which measures how much element i is covered by S, while $\mathcal{C}_i(V)$ is how much element i could possibly be covered (by the whole input). When $\min\{\mathcal{C}_i(S), \alpha \mathcal{C}_i(V)\} = \alpha \mathcal{C}_i(V)$, adding more sentences similar to i will not improve the coverage of the summary, so the algorithm would focus on covering other, thus uncovered, sentences. We note that α is a threshold value, which essentially says that there is not much more value in covering the whole element rather than some portion of it.

They define the diversity metric as

$$\mathcal{R}(S) = \sum_{i=1}^{K} \sqrt{\sum_{j \in P_i \cap S} r_j}$$

Here the input set V is partitioned into K partitions P_1, ..., P_k. The value r_j is the *singleton reward* of j, that is the reward of adding it to an empty summary. Due to the square root, as soon as an element is chosen from a cluster, picking additional elements from that cluster starts to have diminishing returns, and it is better to pick an element from a thus unchosen cluster. The authors propose some extensions to this basic diversity function, such as using another concave-monotonic increasing in place of the square root and using overlapping clusters rather than partitions.

In [16], Kusner et al. introduce the *word mover's distance* (WMD), a novel distance metric between text documents. This metric requires no hyperparameters, and they demonstrate its success through extrinsic evaluation on 8 different k-nearest neighbor classification tasks. Kusner et al. note that prior work have used primarily bag-of-word (BOW) or TF-IDF vectors to represent documents numerically, but these are unsuitable due to their near-orthogonality. Furthermore, they note that these methods usually use exact word match, and don't consider the semantic difference between words. As an example, they give the sentences *Obama greets press in Chicago.* and *President speaks to media at Illinois.*. Figure 2.4 shows a visualization of the word movers' distance between these two sentences. We note the similarity of the embeddings of the words in these two sentences. A simple BOW model would not realize the close semantic relationship between the words in these sentences, but rather only note that they share no words. Kusner et al. note that there is some prior work that seeks to convey semantic information in the representation, but in practice these do not improve performance on distance-based tasks.

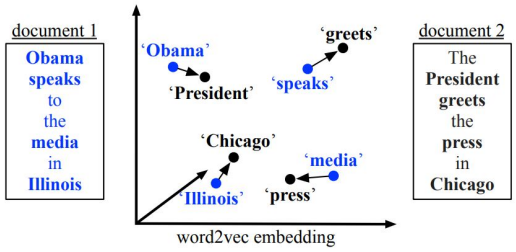

Figure 2.4: Word mover's distance example [16].

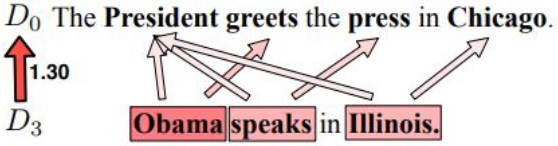

Figure 2.5: Example of the word mover's distance, with overlapping flows.

Kusner et al. represent documents as point clouds of word embeddings, and the distance between documents d and d' as the minimum total distance words from d need to travel to match the point cloud of d'. They note that this is an application of the common earth mover's distance [17]. Further, they note that this metric has a few attractive properties, namely that it is hyperparameter free, easily interpretable, and is able to easily encorporate knowledge from external word embeddings.

Formally, the distance is computed as follows. We are given an embedding matrix $X \in \mathbb{R}^{d \times n}$, with d-dimensional embeddings for a n-word vocabulary. We first represent text documents as normalized BOW vectors (after removing stopwords), $d \in \mathbb{R}^n$, where if word i appears c_i times in a document

$$d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$$

We then note that the word travel cost between words i and j is $c(i,j) = ||x_i - x_j||_2$, where x_i and x_j are the corresponding embeddings from X.

Given two documents representation d and d', we note that any word $i \in d$ can be transformed into any word $i' \in d'$. As such, we construct a flow matrix $T \in \mathbb{R}^{n \times n}$ where $T_{i,i'} \geq 0$ denotes how much of a word $i \in d$ travels to a word $i' \in d'$. We note that if $T_{i,i'} = d_i$ (where d_i is computed as defined above), then all of word $i \in d$ moves to i'. We finally define the distance between the two documents as the minimum weighted (by $T_{i,i'}$) cumulative cost to move all words from d to d': $\sum_{i,i'} T_{i,i'} c(i,i')$. Figure 2.5 shows an example of this flow matrix, with entries being non-zero for the indicated pairs of words. Additionally, we also see that the word movers' distance generalizes to documents of different lengths.

Kusner et al. note that minimizing this cost is equivalent to solving the following optimization problem:

$$\begin{aligned} &\min_{T \geq 0} \sum_{i,i'=1}^{n} T_{i,i'} c(i,i') \\ \text{subject to} &\sum_{i'=1}^{n} T_{i,i'} = d_i && \forall i \in \{1,...,n\} \\ &\sum_{i=1}^{n} T_{i,i'} = d'_{i'} && \forall i' \in \{1,...,n\} \end{aligned}$$

As this a special case of the earth mover's distance, techniques used to solve that problem can be easily adapted to solve the one here as well. Kusner et al. proceed to discuss various optimizations / approximations to make this distance feasible to compute for large corpora. However, as the 'corpora' we are concerned with (meetings are our corpora, and sentences are our documents) are relatively small (around 200 sentences each), we are able to compute the exact word mover's distance.

In [18], Zha considers the problem of generic summarization using sentence clustering. Specifically, they use spectral graph clustering to partition the sentences in the document, and extracts keywords and generates summaries for each group of sentences. They introduce the principle of mutual reinforcement, which dictates that important terms appear in important sentences, and important sentences contain important terms. Specifically, they generate a set of terms (can be some subset of words, all words, phrases, etc.) $T = \{t_1, ..., t_n\}$ and a set of sentences $S = \{s_1, ..., s_m\}$. They create a weighted bipartite graph G with terms T on one side and sentences S on the other, with an edge from t_i to s_j if term t_i appears in sentence s_j. They then define weights on the edges to be the number of times a term appears in a sentence.

Given this graph, they wish to compute *saliency scores* for each term and each sentence, according to the following mutual reinforcement principle: *A term should have a high saliency score if it appears in many sentences with high saliency scores while a sentence should have a high saliency score if it contains many terms with high saliency scores.* Formally, the saliency scores of the terms and sentences are defined as:

$$u(t_i) \propto \sum_{s_j \sim t_i} w_{ij} v(s_j)$$
$$v(s_j) \propto \sum_{t_i \sim s_j} w_{ij} u(t_i)$$

where the $\sim$ operator indicates the presence of an edge between its arguments. The above equations can be expressed in matrix form as

$$u = \frac{1}{\sigma} W v$$
$$v = \frac{1}{\sigma} W^T u$$

The next step Zha takes is to cluster sentences into groups. They do so using a sentence graph, an undirected, weighted graph with sentences as nodes and edges between them representing similarity (or distance). The sentence clustering problem then becomes a graph clustering problem, for which many algorithms exist. Zha notes that documents are essentially sequences of sentences - that is, the sentences are arranged in a linear order, and near-by sentences in this order tend to be about the same topic. This observation yields the *sentence link prior*, which they seek to fully exploit during the clustering process. They note that this is especially useful for a transcript-based summarization task they wish to perform, as the transcripts have no paragraph beginning/ending cues to hint at

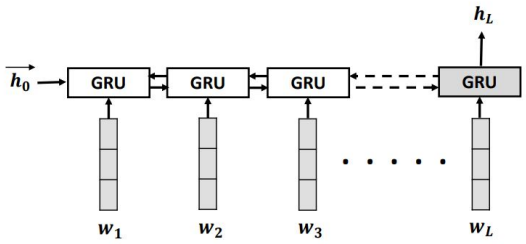

Figure 2.6: Computing sentence embeddings using a GRU.

topic boundaries. Specifically, they proceed to modify the similarity scores of near-by sentences:

$$\hat{w}_{ij} = \begin{cases} w_{ij} + \alpha & \text{if } s_i \text{ and } s_j \text{ are nearby} \\ w_{ij} & \text{otherwise} \end{cases}$$

α is the *sentence link strength*, a tunable parameter signifying how similar nearby sentences are. They also note that the notion of "nearby-ness" can also be tuned, such as considering only consecutive sentences, sentences in a window of two, etc.

Zha proceeds to create a summary hierarchy using this graph. They first make a sentence cluster hierarchy, with lower-level clusters representing finer level details, and higher level clusters representing more high-level ideas, and then create a summary for each node of the hierarchy using sentences belonging to it.

In [19], Nayeem et al. approach the task of abstractive multi-document summarization. They design an unsupervised system, using paraphrastic sentence fusion.

They build upon the work of [2] and [3], and start by constructing a word graph for the sentences they wish to summarize. They note that their approach differs from these works in that it performs lexical paraphrasing, in addition to just deletion-based compression. After generating the K shortest paths, Nayeem et al. seek to re-rank them based on the information they contain. Specifically, they use the TextRank algorithm [5], and create an undirected graph where candidates are vertices, and weighted edges connect similar candidates. They note that the original algorithm uses textual overlap, so if two sentences discuss the same topic using different words, there would be no edge between them. To remedy this, Nayeem et al. use sentence embeddings. Using pretrained word embeddings, they use a bi-directional GRU and obtain sentence embeddings by concatenating forward and backward hidden states. This process is demonstrated in Figure 2.6, in which words w_1, ..., w_L are fed in and the embedding h_L is obtained. They use cosine similarity between these embeddings to create edges in the sentence similarity graph. They also use these word embeddings to perform context-sensitive word substitutions.

They note that the above summarization algorithm is applied on a subset of sentences at a time, specifically ones that have been identified as sharing information. They use hierarchical agglomerative clustering [20] with a complete linkage criteria. In this method, each sentence initially starts off in its own cluster, and pairs of similar clusters are merged at each step. The complete linkage criteria specifies clusters c and c' are merged if and only if the largest distance between a sentence in c and a sentence in c' is the smallest among any pair of clusters. They measure sentence distance using cosine similarity between sentence embeddings (computed using the GRU as described above). They stop merging clusters when all pairs of

clusters are separated by a certain distance, a tunable hyperparameter. They note that clusters resulting from this algorithm may be small, but have the property that any two sentences in a cluster are similar to each other, not merely transitively similar. They then choose a single sentence from each cluster, which both reduces redundancy (since the clusters are similar) and increases diversity (since sentences from different clusters are chosen).

After generating a sentence for each cluster of related sentence, they pick the sentences in the final summary to output. They use a concept-based integer linear programming (ILP) framework, seeking to extract sentences covering as many important concepts as possible, while limiting summary length to a given budget. They use keyphrases (words or phrases representing the main topics of a document) as concepts, and choose sentences that contain many key phrases. Formally, their algorithm proceeds as follows. Letting w_i be the weight of keyphrase i, k_i be an indicator variable for whether the keyphrase i appears in the selected subset, l_j be the number of words in sentence j, s_j be an indicator variable for whether the sentence j appears in the selected subset, $occ_{i,j}$ indicate the occurrence of keyphrase i in sentence j, and L be the length budget, they formulate the integer program:

$$\max \sum_i w_i k_i + \sum_j s_j \left(\text{score}(s_j) + \frac{l_j}{L} \right) \text{ [maximize keyphrase weight and model score]}$$

$$\text{subject to} \sum_j l_j s_j \leq L \text{ [total length limit]}$$

$$s_j occ_{i,j} \leq k_i, \forall i, j$$

[the number of keyphrase occurrences is number of sentence occurrences containing it]

$$\sum_j s_j occ_{i,j} \geq k_i, \forall i$$

[the number of keyphrase occurrences is number of sentence occurrences containing it]

$$\sum_{j \in g_c} s_j \leq 1$$

[use at most one sentence from a cluster]

$$k_i \in \{0, 1\}$$

$$s_j \in \{0, 1\}$$

CHAPTER 3

CONTRIBUTIONS

3.1 Original System

In [1], Shang et al. approach the task of abstractive meeting summarization using multi-sentence compression and submodular optimization. Specifically, their system does not rely on any annotations for the input data, and the system is fully unsupervised end-to-end. Their system is composed of four main components, the first to preprocess input text, then one to group related sentences into *communities*, then one to make an abstractive sentence for each community, and finally one to select a subset of the abstractive sentences.

The text preprocessing component is fairly standard. Some notable additions stem from the fact that this system is working on automated speech recognition (ASR) output, and as such Shang et al. remove repeated unigrams and bigrams, filter out specific ASR tags corresponding to gaps and filler words. They also filter out consecutive stopwords at the ends of the sentence, and remove sentences with fewer than 3 non-stopwords.

The next step of their pipeline is the community detection component, which is tasked with grouping together utterances that should be summarized by a single abstractive sentence. They assign TF-IDF weights to each of the utterances, and use Latent Semantic Analysis (LSA) to reduce the dimensionality of the resulting matrix. They note that LSA is used here to generate a stable clustering, that is

resilient to noise (small changes in the input utterances).

After community detection, they proceed to make a summary sentence per community, which is generated independently for each community. Following the steps of [6], word importance scores are first computed using the CoreRank algorithm. These scores are then reweighted using a metric similar to Inverse Document Frequency (IDF), using each community as a document and the whole meeting as the collection. Shang et al. note that this is a reasonable reweighting scheme, as terms are considered important if they are both important in their own community (have a high CoreRank score) and appear in a relatively small number of communities (IDF score). They refer to these scores as the TW-IDF scores for each node.

They next generate a word graph, building on the work of [2] and [3]. They use Filippova's edge weights (Equation 2.1), with one modification: they include a term $d^2_{p_i,p_j}$, which is the Euclidean distance between the embeddings of words mapped to nodes p_i and p_j in the graph. They note that this favors paths that pass through salient words that are highly similar to one another. This helps ensure that the path doesn't pass through completely unrelated words. Their final edge weights are:

$$w(e_{i,j}) = \frac{(\text{freq}(i) + \text{freq}(j)) \times d^2_{p_i,p_j}}{\left(\sum_{s \in S} \text{diff}(s,i,j)\right) \times (\text{freq}(i) \cdot \text{freq}(j))} \tag{3.1}$$

Building on the work of [3], they then find the K shortest paths in the word

graph, and filter out paths with less than z words. Next, they cluster all the words in the MSCG using k-means. They then consider a few metrics to rerank these paths, such as fluency ($F(P)$, measured using a trigram model), coverage ($C(P)$, measured as the sum of the TW-IDF scores of all nodes in the path as computed above), and diversity ($D(P)$, measured as the number of clusters the path passes through):

$$F(P) = \frac{\sum_{i=1}^{|P|} \log Pr(p_i|p_{i-n+1}^{i-1})}{\text{number of n-grams in } P}$$
$$C(P) = \frac{\sum_{p_i \in P} \text{TW-IDF}(p_i)}{\text{number of nouns, verbs, and adjectives in P}}$$
$$D(P) = \frac{\sum_{j=1}^{k} 1_{\exists p_i \in P | p_i \in \text{cluster}_j}}{|P|}$$

They then rerank the paths according to a novel weighting scheme combining these three metrics (with lower weights being better):

$$\text{score}(P) = \frac{W(P)}{|P| \times F(P) \times C(P) \times D(P)}$$

The sentences generated by the MSC step can already be considered an abstractive summary. However, in order to allow their system to generate summaries meeting various requirements (eg. maximum length, etc.) and remove redundancy, the

final step in their pipeline is budgeted submodular optimization. They define the following submodular function:

$$f(S) = \sum_{s_i \in S} n_{s_i} w_{s_i} + \lambda \sum_{j=1}^{k} 1_{\exists s_i \in S | s_i \in \text{cluster}_j}$$

where s_i is a word, n_{s_i} is the number of occurrences of that word in S, and w_{s_i} is the CoreRank score of s_i. CoreRank scores and clusters are computed similarly to before, except now this is done on the whole summary rather than an individual community.

3.2 Modifications

We investigate modifications to the community detection step of the pipeline. Specifically, we propose using an embedding-based distance, in order to capture word similarity. We note that if semantically-similar words are used in a meeting, the current system has no way of detecting their similarity, and as such will not be able to create an accurate distance. Furthermore, using word embeddings will allow us to use external information, which we hypothesize will be beneficial due to the limited data present for each meeting.

We use the word-movers' distance, proposed in [16]. Specifically, we run the WMD

algorithm on each meeting separately, representing utterances as documents and a single meeting as a corpus. We initially used a custom implementation of the word movers' distance, but ultimately adapted one from Gensim [21] due to the speed provided by their optimized code. As the first step in our community detection, we computed an $n_m \times n_m$ distance matrix for each meeting m (where n_m represents the number of utterances in m).

Additionally, we also investigated the impact of providing sentence-proximity information to the community detection (similar to the work of [18]). This was done by reweighting the distances computed by the WMD step as follows:

$$d(s_i, s_j) = \begin{cases} \frac{d(s_i,s_j)}{2} & |i-j| = 1 \\ d(s_i, s_j) & \text{otherwise} \end{cases} \tag{3.2}$$

Specifically, we reduced the distance of consecutive sentences by a factor of two, and left other distances unchanged. Note that we are using a multiplicative reweighting scheme rather than the additive one found in [18]. We expect that since [18] was reweighting similarity scores while we are reweighting distances, a multiplicative scheme works better here.

After generating these distance matrices, we sought to create a partition of the utterances into communities. We tried three clustering algorithms for this step, namely agglomerative clustering, spectral clustering, and k-means clustering on

projected data points. For agglomerative clustering, we chose to use complete-link variant (similar to the work of [19]). We follow the steps of [18] to perform spectral clustering.

To use k-means for this task, we first need to project the distance matrix onto a vector space, and then run k-means on these projected points. This is done using the Multidimensional Scaling (MDS) algorithm [22]. The MDS algorithm is quite involved and out of the scope of this work, but it is essentially rearranging n objects in a d-dimensional space in such a way that best preserves given pairwise distances between the objects. We project the points onto a 6-dimensional space. We note that even if the true representation of these points lies in a higher N dimensional space $\mathbb{R}^N$, it is valid to project them to a smaller n dimensional space $\mathbb{R}^n$ due to the Johnson-Lindenstrauss lemma [23], which states:

> Given $0 \leq \epsilon \leq 1$, a set X of m points in $\mathbb{R}^N$ and a number $n > \frac{8\ln(m)}{\epsilon^2}$, there is a linear map $f : \mathbb{R}^N \to \mathbb{R}^n$ such that

$$(1-\epsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\epsilon)||u-v||^2$$

As the meetings have less than $m = 200$ points, we need to pick $n = O(\ln(200)) \approx 5.29 \leq 6$.

After running one of these six systems (three clustering methods on two distance metrics), the resulting communities were provided to the multi-sentence compression stage of the algorithm, and the last two steps of the pipeline were run.

CHAPTER 4

EXPERIMENTS

4.1 Dataset

We conducted experiments on the commonly used AMI Meeting Corpus [24]. This corpus contains data from 100 hours of meeting recordings, composed from a total of 67 meetings. These meetings are further split into a training set of 47 meetings and a testing set of 20 meetings. Both human-generated and ASR-generated transcripts are provided for these meetings; in order to create a fully-unsupervised meeting we use the ASR-generated transcripts. We note that the word error rate of the ASR transcription is 36%. Each meeting also comes with a human-generated reference summary of on average 290 words.

4.2 Baselines

We compare our system against 10 baselines. Four of these are for the community detection stage specifically:

- Random - utterances are randomly split into communities
- Sequential - each meeting is divided into equal parts, and a community is created from each part
- Single - a single community is created containing all utterances

- Individual - each utterance is assigned to its own community

We also compared our systems to the system created by Shang et al. [1].

4.3 Our systems

We ran six of our own systems. We defined a system based on a combination of distance metric and a clustering method. We chose one of the following distance metrics:

- Word movers' distance
- Word movers' distance with reweighting as in 3.2

and one of the following clustering methods:

- Agglomerative clustering
- Spectral clustering
- k-means clustering on embeddings computed using multidimensional scaling on distances

4.4 Hyperparameters

For each of our 6 systems described in Section 4.3, we conducted a grid search using the AMI development set, for a fixed summary size of 350 words. Namely, we searched over the following parameters:

- Minimum path length (z) - searched over values in the range $[6, 16]$ with a step size of 2. This parameter is necessary because while we want short paths, paths that are too short will likely correspond to invalid sentences, and thus should be pruned out.
- Number of communities (n) - fixed at 50. This parameter essentially controls the abstractiveness of our summary. If it is set to a large value, then all utterances will be assigned to their own community, causing our MSC to just pick each utterance, and creating an extractive summary. Lower values correspond to more abstractive summaries. We note that Shang et al. realized optimal performance for almost all their systems (3/4) with $n = 50$ (the last system, described above as Their system (FluCovRank) performs best with $n = 35$). As such, we only include $n = 50$ in our grid to significantly decrease the computational cost of the search. We note that searching over other values of n could lead to improved performance.
- Number of diversity clusters (k) - this value is tied to the minimum path length z for the grid search. Shang et al. did not provide justification for why this was done, but we follow the same procedure to maintain comparability of the two systems. This parameter is used in the path reranking step, where we count how many clusters of the word graph a given path hits to estimate

its 'diversity'. Note that smaller values will lead to fewer clusters and less granularity while larger values will lead to more clusters and a more granular estimate.

- CoreRank window size (w) - searched over $\{6, 12\}$. This parameter dictates the window size used for the CoreRank algorithm. Note that high values will lead to a dense graph of words while low ones will give rise to a sparser graph.
- CoreRank overspanning flag - searched over {`TRUE, FALSE`}. This parameter detects whether words within a window from separate sentences are considered neighbors (and thus give rise to edges in the graph of words).
- Submodularity parameters (tradeoff parameter λ, scaling factor r) - fixed at $\lambda = 0.7, r = 0.5$. These parameters control the submodular optimization, specifically the tradeoff between coverage (total weighted CoreRank score) and diversity (number of clusters the summary covers). We hypothesize that the optimal values for these hyperameters for our system will be similar to the ones obtained by Shang et al. This is due to the modular design of the system, and the fact that we are leaving the MSC portion of the pipeline between community detection and submodular optimization unchanged. This is done in order to reduce the amount of computation needed - we note that searching over other values of λ, r could lead to improved performance.

The optimal hyperparameter values obtained are described in Table 4.1. In addition to these parameters, we set K, the number of shortest paths extracted from the MSCG to 200 following the setup of [1]. They note that increasing K from 100 provides diminishing returns on performance, and significantly increases computational cost. The value of $K = 200$ was chosen empirically as a tradeoff between

Table 4.1: Optimal parameter values

System	n	w	z	Overspanning	λ	r
WMD-AGG	50	6	14	false	0.7	0.5
WMD-SPEC	50	6	12	true	0.7	0.5
WMD-kMEANS	50	12	14	false	0.7	0.5
WMDr-AGG	50	6	16	false	0.7	0.5
WMDr-SPEC	50	6	8	false	0.7	0.5
WMDr-kMEANS	50	10	12	true	0.7	0.5

the two.

After the grid search, we find the following optimal parameter values for our systems, which are given in Table 4.1.

CHAPTER 5

RESULTS

5.1 Metrics

We use the ROUGE-1, ROUGE-2, and ROUGE-SU4 metrics [27] to evaluate performance, in line with prior work for ease of comparison. These metrics are based on unigram, bigram, and unigram plus skip-bigram overlap (with max skip distance of 4). We note that for all the systems listed here (our systems, the baselines, as well as Shang et al.'s system), the summaries are generated based on ASR transcripts and compared against human-generated abstractive summaries. Since ROUGE is based on token overlap, it is hard to achieve very high scores since many tokens in the ground truth summaries don't occur in the meeting transcript at all.

5.2 Evaluation

The results for our baselines, Shang et al.'s system ("Their System"), and our systems are shown in Table 5.1.

Table 5.1: Macro-averaged results for 350 word summaries.

	ROUGE-1			ROUGE-2			ROUGE-SU4		
	R	P	F-1	R	P	F-1	R	P	F-1
WMD-AGG	40.68	33.61	36.32	7.64	6.40	6.87	15.31	13.19	14.00
WMDr-AGG	40.85	**33.80**	**36.49**	7.65	6.39	6.87	15.27	12.99	13.86
WMD-SPEC	**40.94**	33.66	36.43	**7.94**	**6.60**	**7.10**	**15.53**	**13.54**	**14.28**
WMDr-SPEC	40.68	33.47	36.22	7.60	6.44	6.87	14.86	13.29	13.84
WMD-kMEANS	40.32	33.33	35.99	7.43	6.23	6.69	15.26	12.96	13.86
WMDr-kMEANS	40.37	33.47	36.11	7.33	6.35	6.72	15.23	13.36	14.06
Sequential Baseline	38.50	35.13	35.86	7.39	6.80	6.92	14.55	13.67	13.79
Random Baseline	38.13	33.36	35.00	6.52	5.77	6.04	14.14	12.68	13.17
Single Baseline	3.76	**56.68**	7.00	0.82	**13.29**	1.53	1.54	**23.83**	2.88
Individual Baseline	39.82	32.71	35.42	8.16	6.89	**7.37**	15.09	13.48	14.04
Their System	41.83	34.44	37.25	8.22	6.95	7.43	15.83	13.70	14.51

5.2.1 ROUGE-1

Our systems all outperform the baselines in terms of F1 score. We note that the *Single* baseline, in which a single community is created containing all the utterances, does poorly in terms of recall, while having very good precision. We hypothesize that this is because splitting the data into separate communities (which is done by all the other systems) encourages the model to pick a variety of topics which results in higher precision. However, since this approach requires taking only one summary sentence from each topic, the model may thus under-represent important topics resulting in lower recall. Among our models, WMDr-AGG performs the best, although it is still beat by Shang et al.'s system.

5.2.2 ROUGE-2

Our system outperforms all baselines in terms of F1 score, except for the *Individual* baseline. We note that this is very reasonable. In this baseline, each utterance is

in its own "community", so the MSC step does not actually perform compression and individual utterances are preserved. Furthermore, during the submodular optimization step no summaries are modified, and thus utterances in the input are preserved in the final summary. As such, more bigrams present in the reference summary are found in the one produced by the *Individual* baseline than the other systems.

Similar to the ROUGE-1 results, the *Single* baseline outperforms our systems in terms of precision, while it suffers from poor recall. We expect that the reasons for this are the same as the ones outlined above. Among our models, WMD-SPEC performs the best, although it is still beat by Shang et al.'s system.

5.2.3 ROUGE-SU4

Similar to ROUGE-2, our system outperforms all baselines in terms of F1 score, except for the *Individual* baseline. We expect that this is due to a similar reason as for ROUGE-2, namely that since each community contains only one utterance, the MSC step does not modify utterances and the submodular optimization is run on utterances from the meeting. The *Single* baseline also has better precision than our system, at the cost of much lower recall. WMD-SPEC performs the best among our model, although it still does not meet the performance of Shang et al.'s system.

CHAPTER 6

CONCLUSION

In this work, we examined the performance of various community detection algorithms when applied to meeting summarization. We were able to beat our baselines for almost all of the metrics we considered, except for a few cases in which the reason for high baseline performance is clear. Specifically, we note that our WMD-SPEC system performed quite well across the metrics, getting both optimal precision and recall (and thus optimal F-1 score) in ROUGE-2 and ROUGE-SU4, and has optimal recall on ROUGE-1 as well. It is marginally beaten by WMDr-AGG on ROUGE-1 precision and F-1 score.

For future work, it would be interesting to consider other clustering methods, as well as other distance metrics. Specifically, we would like to consider distance metrics more suited to sentence fragments rather than whole sentences. This is because due to the nature of the data being worked with (coming from ASR transcripts), sometimes a single sentence gets split into multiple utterances. Additionally, we would like to make better use of the speaker tag (ID of the person speaking) for each utterance. This could be used in several different ways, such as hypothesizing that the same person would not say the same thing twice, and thus their utterances should all be in distinct communities, or that the same person would be interested in the same topics, and as such their utterances should clustered together. Additionally, our reweighting scheme only modified the distances between neighboring utterances, future work could modify the distances for more pairs of utterances based on proximity, including even all the utterances using a decaying weight scheme.

Future work could also consider deep learning based approaches. However, we note that there is relatively little labeled data for meeting summarization, which may cause problems with a neural approach.

BIBLIOGRAPHY

[1] Guokan Shang, Wensi Ding, Zekun Zhang, Antoine Jean-Pierre Tixier, Polykarpos Meladianos, Michalis Vazirgiannis, and Jean-Pierre Lorré. Unsupervised abstractive meeting summarization with multi-sentence compression and budgeted submodular maximization. *arXiv preprint arXiv:1805.05271*, 2018.

[2] Katja Filippova. Multi-sentence compression: Finding shortest paths in word graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 322–330. Association for Computational Linguistics, 2010.

[3] Florian Boudin and Emmanuel Morin. Keyphrase extraction for n-best reranking in multi-sentence compression. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2013.

[4] Xiaojun Wan and Jianguo Xiao. Single document keyphrase extraction using neighborhood knowledge. In *AAAI*, volume 8, pages 855–860, 2008.

[5] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[6] Antoine Tixier, Fragkiskos Malliaros, and Michalis Vazirgiannis. A graph degeneracy-based approach to keyword extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1860–1870, 2016.

[7] Marina Litvak and Mark Last. Graph-based keyword extraction for single-document summarization. In *Proceedings of the workshop on Multi-source*

Multilingual Information Extraction and Summarization, pages 17–24. Association for Computational Linguistics, 2008.

[8] Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16:3–1, 2008.

[9] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888, 2010.

[10] Yashar Mehdad, Giuseppe Carenini, Frank Tompa, et al. Abstractive meeting summarization with entailment and fusion. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 136–146, 2013.

[11] Gabriel Murray, Giuseppe Carenini, and Raymond Ng. Using the omega index for evaluating abstractive community detection. In *Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*, pages 10–18. Association for Computational Linguistics, 2012.

[12] Steve Gregory. An algorithm to find overlapping community structure in networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 91–102. Springer, 2007.

[13] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

[14] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[15] Jaime G Carbonell and Jade Goldstein. The use of mmr and diversity-based reranking for reodering documents and producing summaries. 1998.

[16] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[17] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

[18] Hongyuan Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 113–120. ACM, 2002.

[19] Mir Tafseer Nayeem, Tanvir Ahmed Fuad, and Yllias Chali. Abstractive unsupervised multi-document summarization using paraphrastic sentence fusion. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1191–1204, 2018.

[20] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295, 2014.

[21] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[22] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. Chapman and hall/CRC, 2000.

[23] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[24] Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, et al. The ami meeting corpus: A pre-announcement. In *International Workshop on Machine Learning for Multimodal Interaction*, pages 28–39. Springer, 2005.

[25] Korbinian Riedhammer, Dan Gillick, Benoit Favre, and Dilek Hakkani-Tür. Packing the meeting summarization knapsack. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[26] Nikhil Garg, Benoit Favre, Korbinian Reidhammer, and Dilek Hakkani-Tür. Clusterrank: a graph based method for meeting summarization. In *Tenth Annual Conference of the International Speech Communication Association*, 2009.

[27] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.